Game Development for iPhone

Part 2

Session 305

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

Game Development for iPhone

Part 2

Allan Schaffer
Graphics Technology Evangelist

Agenda

- Part 1
  - Welcome and Overview — Done
  - Graphics — Done
  - Audio — Done
- Part 2
  - Peer-to-Peer Connectivity — 20 minutes
  - Accelerometer — 20 minutes
  - In-App Purchase — 20 minutes

Agenda

- Part 1
  - Welcome and Overview — Done
  - Graphics — Done
  - Audio — Done
- Part 2
  - Peer-to-Peer Connectivity — 20 minutes
  - Accelerometer — 20 minutes
  - In-App Purchase — 20 minutes

Peer to Peer Connectivity

Game Kit Framework

High-level APIs to support networked gaming

Session

- Data communication over Bluetooth
- No pairing needed

Peer Picker

- Standard UI for discovering players

In-Game Voice

- Voice communication over online network
- Your connection

Game Kit Framework

High-level APIs to support networked gaming

Session

- Data communication over Bluetooth
- No pairing needed

Peer Picker

- Standard UI for discovering players

GKSession

Provides connection and data communication

- Handles sending and receiving data
- Client/server or peer-to-peer
- Tracks connection state
- Objective-C encapsulation built atop Bonjour

GKSession

Provides connection and data communication

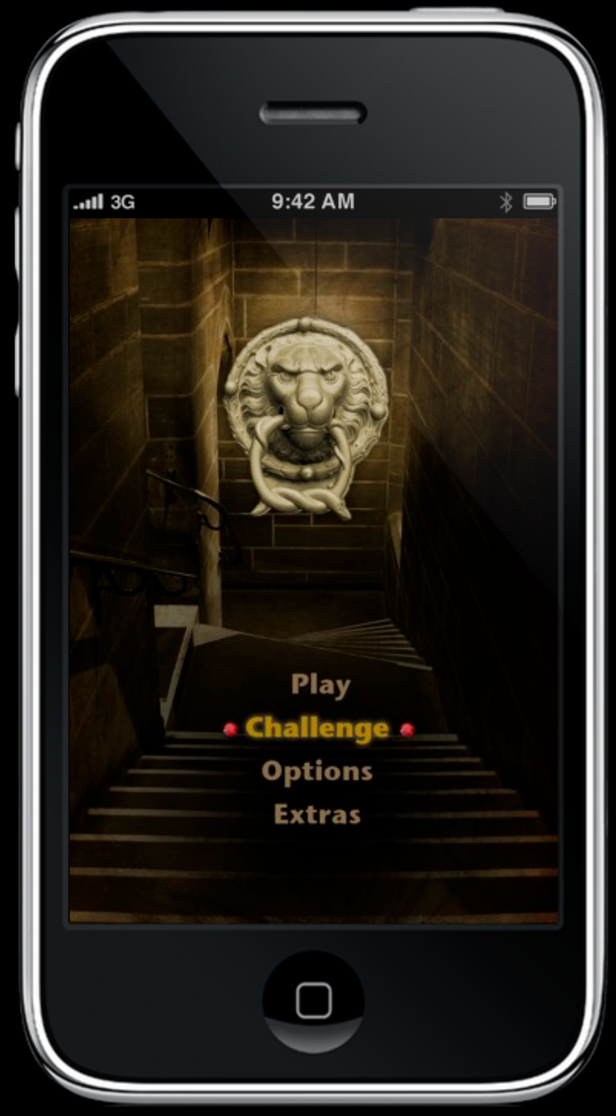
- Handles sending and receiving data
- Client/server or peer-to-peer
- Tracks connection state
- Objective-C encapsulation built atop Bonjour

GKPeerPickerController

Standard UI to initiate a GKSession

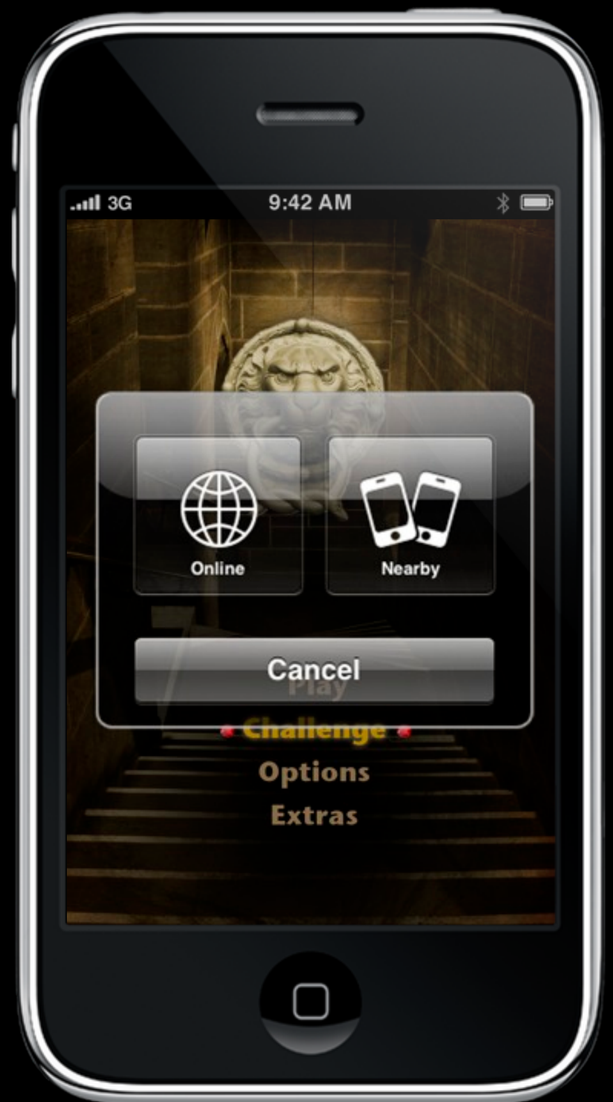
Peer Picker

Control flow



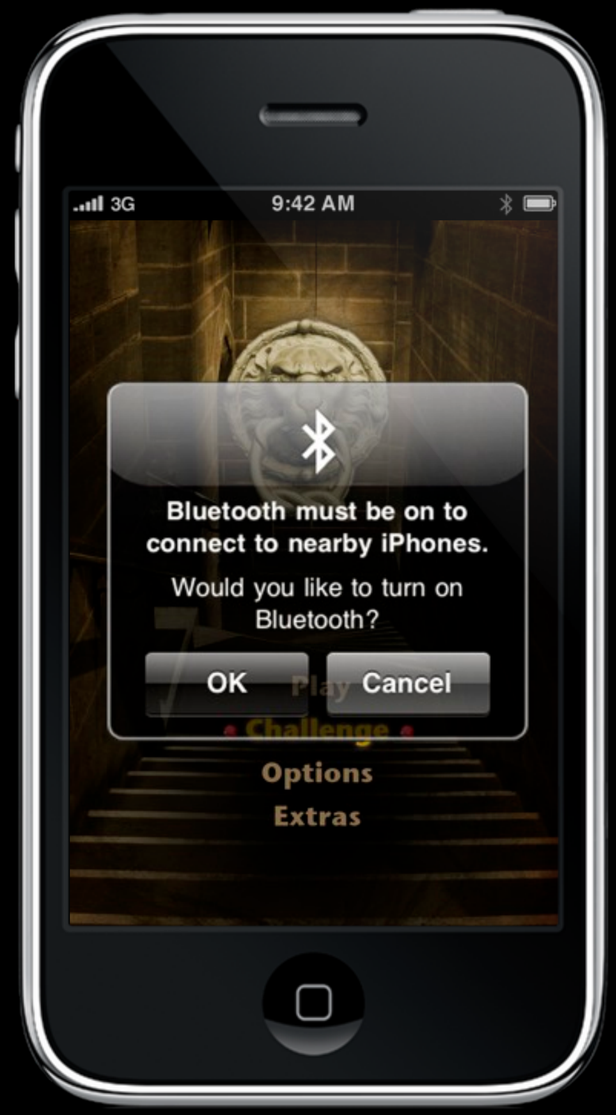
Peer Picker

Choosing connection type



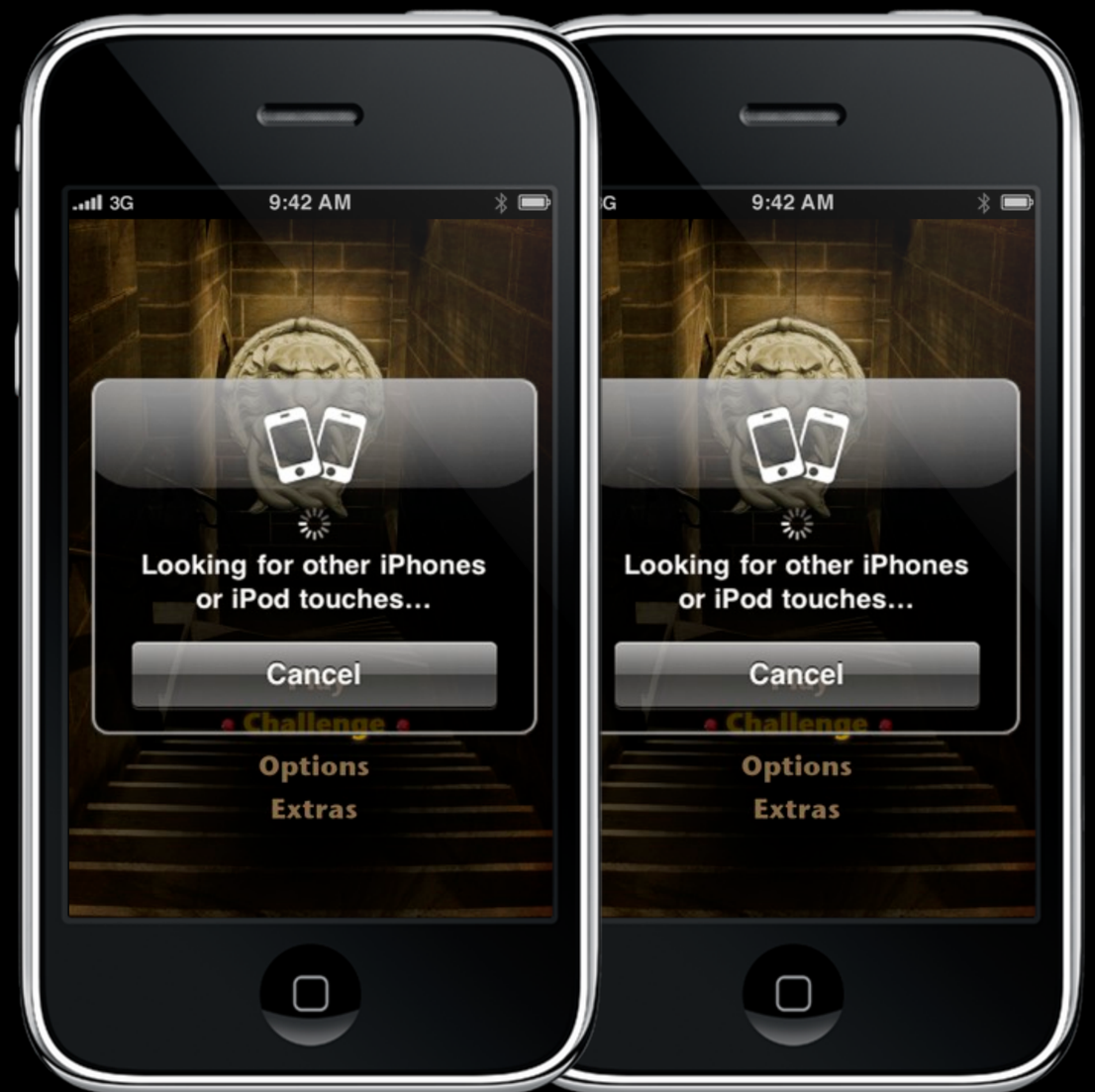
Peer Picker

Enabling bluetooth



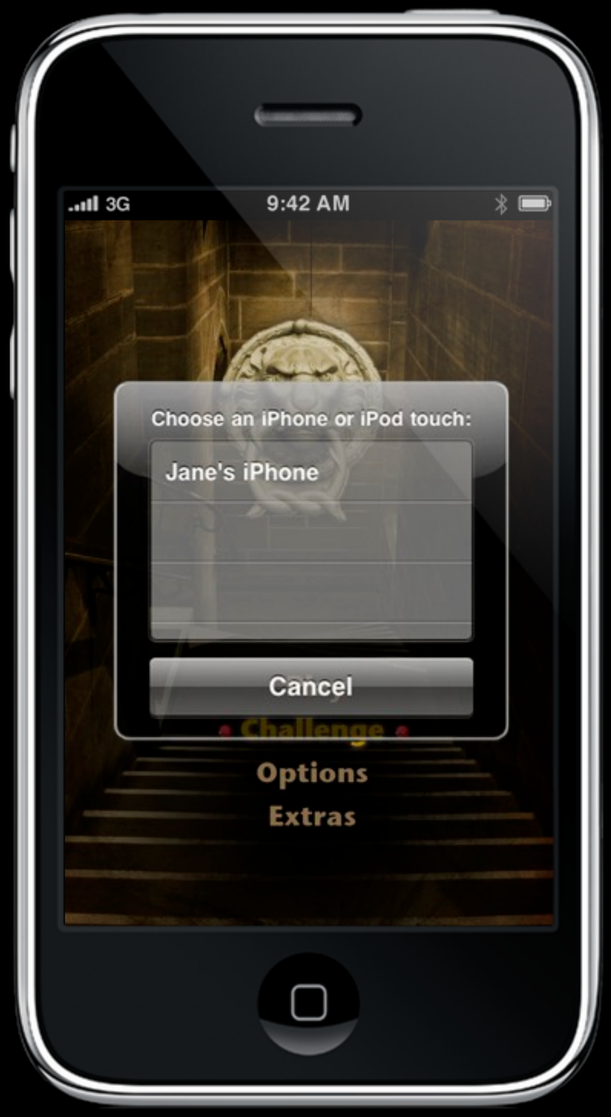
Peer Picker

Discovering peers



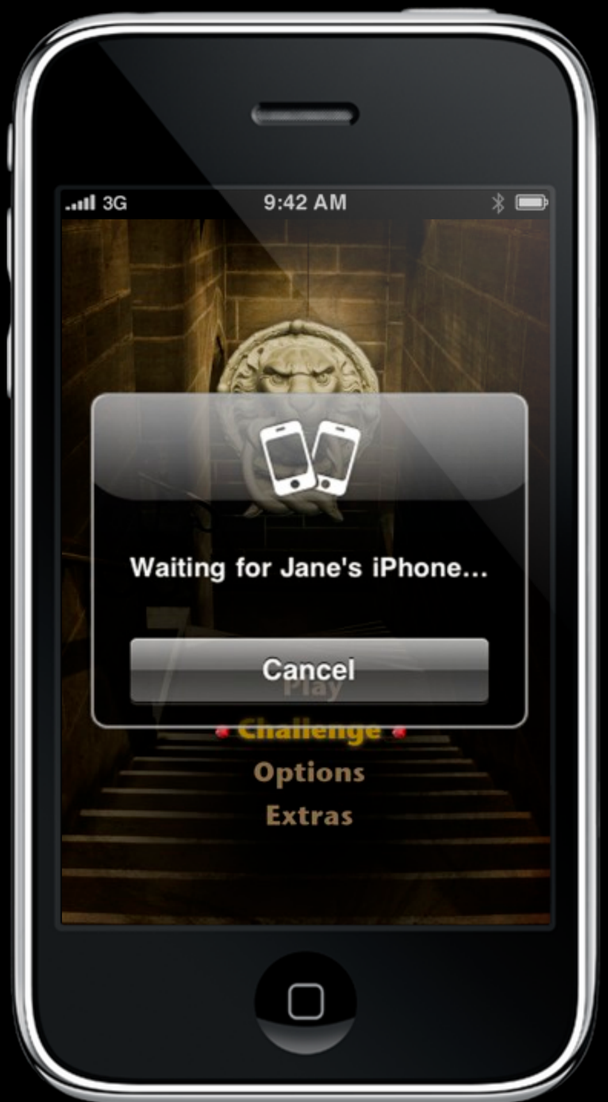
Peer Picker

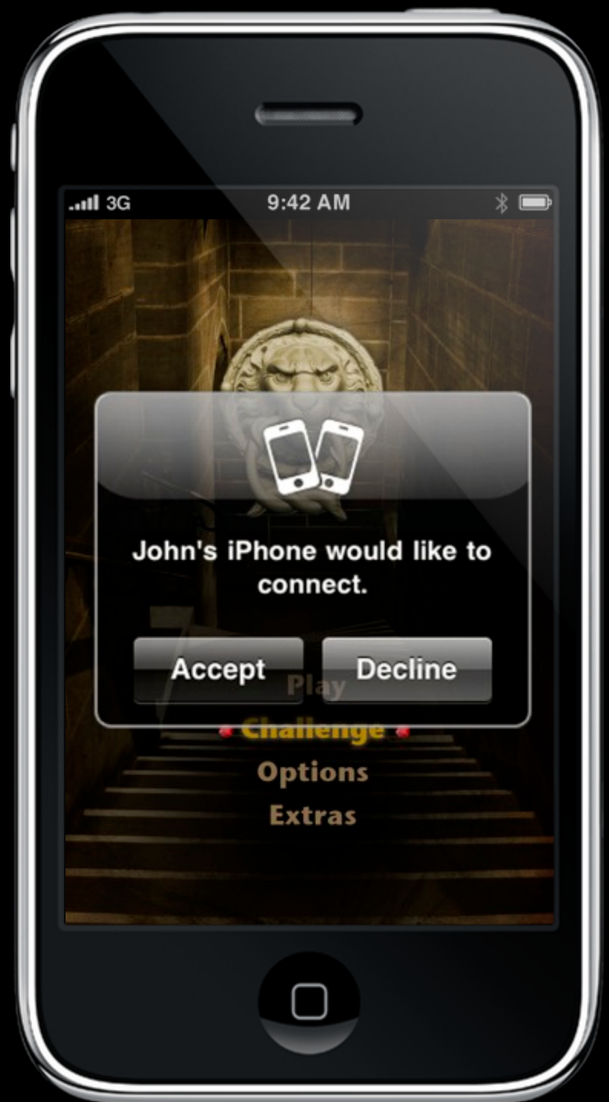
Choosing a peer



Picker Control Flow

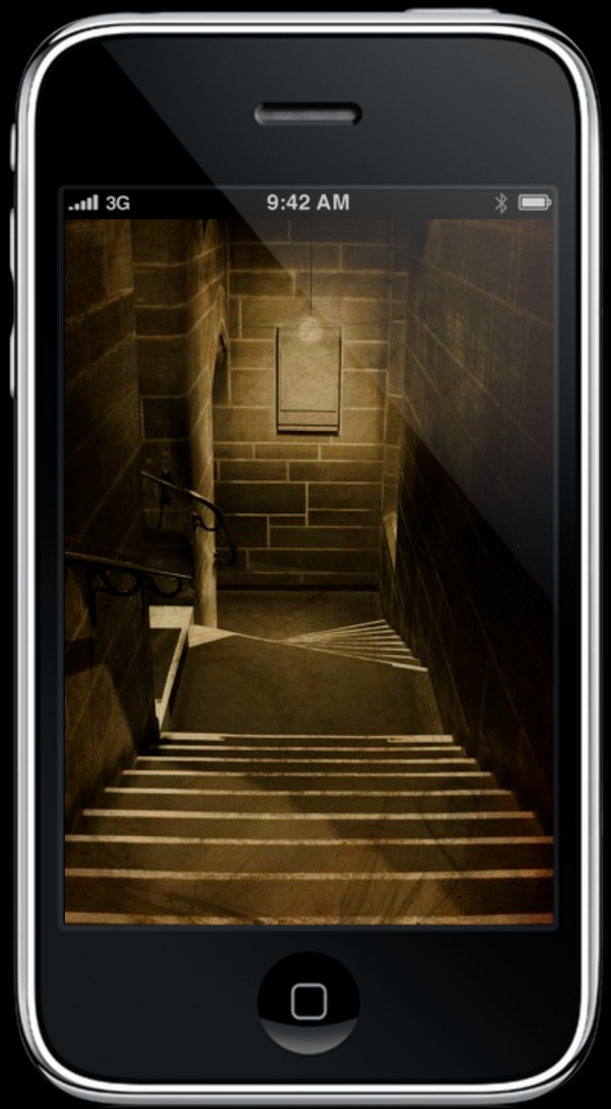
Requesting connection





Picker Control Flow

Connected!



Peer-to-Peer Connection

Steps to connect over bluetooth

1. Configure and present GKPeerPickerController
2. Provide a GKSession

   `-peerPickerController:sessionForConnectionType:`

3. Handle picker connecting to peer (or user cancel)

   `-peerPickerController:didConnectPeer:toSession:`

   `-peerPickerController:didCancel:`

4. Implement methods for GKSession to send and receive data

   `-mySendData:usingDataMode:`

   `-recieveData:fromPeer:inSession:context:`

5. Handle GKSession state changes

   `-session:peer:didChangeState:`

Peer-to-Peer Connection

1. Configure and present GKPeerPickerController

```
-(void)setupPicker {

    // Create the PickerController object
    peerPicker = [ [ GKPeerPickerController alloc ] init ];

    // set delegate
    peerPicker.delegate = self;

    // Configure a connection type (bluetooth)
    peerPicker.connectionTypesMask = GKPeerPickerConnectionTypeNearby;

    // Present picker to the user
    [ peerPicker show ];

}
```

Peer-to-Peer Connection

1. Configure and present GKPeerPickerController

```
-(void)setupPicker {

    // Create the PickerController object
    peerPicker = [ [ GKPeerPickerController alloc ] init ];

    // set delegate
    peerPicker.delegate = self;

    // Configure a connection type (bluetooth)
    peerPicker.connectionTypesMask = GKPeerPickerConnectionTypeNearby;

    // Present picker to the user
    [ peerPicker show ];

}
```

Peer-to-Peer Connection

2. Provide a GKSession

```
// Picker needs you to create and return a GKSession
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
           sessionForConnectionType:(GKPeerPickerConnectionType)type
{
    // Create a session.
    // ID can be a registered bonjour service type (RFC 2782)
    GKSession *session = [[ GKSession alloc ] initWithSessionID:kGameIdentifier
                                                    displayName:kPlayerName
                                                    sessionMode:GKSessionModePeer];

    return [ session autorelease ];
}
```

Peer-to-Peer Connection

2. Provide a GKSession

```
// Picker needs you to create and return a GKSession
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
           sessionForConnectionType:(GKPeerPickerConnectionType)type
{
    // Create a session.
    // ID can be a registered bonjour service type (RFC 2782)
    GKSession *session = [[ GKSession alloc ] initWithSessionID:kGameIdentifier
                                                    displayName:kPlayerName
                                                    sessionMode:GKSessionModePeer];

    return [ session autorelease ];
}
```

http://www.dns-sd.org/ServiceTypes.html

Peer-to-Peer Connection

3. Handle picker connecting to peer

```
- (void)peerPickerController:(GKPeerPickerController *)picker
              didConnectPeer:(NSString *)peerID toSession:(GKSession *)session;
{
    // retain a reference to the session
    self.mySession = session;      // retaining property

    // Set session delegate
    self.mySession.delegate = self;

    // Set delegate for receiving data (step 4)
    [ self.mySession setDataReceiveHandler:self withContext:NULL ];

    // Dismiss & release the picker
    [ picker dismiss ];
    picker.delegate = nil;
    [ picker autorelease ];

    // Connection made!  Start game.
}
```

Peer-to-Peer Connection

3. Handle picker connecting to peer

```
- (void)peerPickerController:(GKPeerPickerController *)picker
              didConnectPeer:(NSString *)peerID toSession:(GKSession *)session;
{
    // retain a reference to the session
    self.mySession = session;      // retaining property

    // Set session delegate
    self.mySession.delegate = self;

    // Set delegate for receiving data (step 4)
    [ self.mySession setDataReceiveHandler:self withContext:NULL ];

    // Dismiss & release the picker
    [ picker dismiss ];
    picker.delegate = nil;
    [ picker autorelease ];

    // Connection made!  Start game.
}
```

Peer-to-Peer Connection

3. Handle picker connecting to peer

```
- (void)peerPickerController:(GKPeerPickerController *)picker
              didConnectPeer:(NSString *)peerID toSession:(GKSession *)session;
{
    // retain a reference to the session
    self.mySession = session;      // retaining property

    // Set session delegate
    self.mySession.delegate = self;

    // Set delegate for receiving data (step 4)
    [ self.mySession setDataReceiveHandler:self withContext:NULL ];

    // Dismiss & release the picker
    [ picker dismiss ];
    picker.delegate = nil;
    [ picker autorelease ];

    // Connection made!  Start game.
}
```

Peer-to-Peer Connection

3. Handle picker connecting to peer

```
- (void)peerPickerController:(GKPeerPickerController *)picker
              didConnectPeer:(NSString *)peerID toSession:(GKSession *)session;
{
    // retain a reference to the session
    self.mySession = session;      // retaining property

    // Set session delegate
    self.mySession.delegate = self;

    // Set delegate for receiving data (step 4)
    [ self.mySession setDataReceiveHandler:self withContext:NULL ];

    // Dismiss & release the picker
    [ picker dismiss ];
    picker.delegate = nil;
    [ picker autorelease ];

    // Connection made!  Start game.
}
```

Peer-to-Peer Connection

3. Handle picker connecting to peer

```
- (void)peerPickerController:(GKPeerPickerController *)picker
              didConnectPeer:(NSString *)peerID toSession:(GKSession *)session;
{
    // retain a reference to the session
    self.mySession = session;      // retaining property

    // Set session delegate
    self.mySession.delegate = self;

    // Set delegate for receiving data (step 4)
    [ self.mySession setDataReceiveHandler:self withContext:NULL ];

    // Dismiss & release the picker
    [ picker dismiss ];
    picker.delegate = nil;
    [ picker autorelease ];

    // Connection made!  Start game.
}
```

Peer-to-Peer Connection

3. (…or user cancel)

```
- (void)peerPickerControllerDidCancel:(GKPeerPickerController *)picker
{
    // tear down picker

    // tear down session

    // return to previous UI
}
```

Peer-to-Peer Connection

4. Implement GKSession methods to send and receive data

```
- (void)mySendData:(NSData *)data
     usingDataMode:(GKSendDataMode)mode
{
    [ self.mySession sendDataToAllPeers:data          // NSData *
                           withDataMode:mode          // reliable or unreliable
                                  error:nil ];
}
```

```
- (void)receiveData:(NSData *)data
           fromPeer:(NSString *)peer
          inSession:(GKSession *)session
            context:(void *)context
{
    // unpack the NSData, update game state
}
```

Peer-to-Peer Connection

4. Implement GKSession methods to send and receive data

```
- (void)mySendData:(NSData *)data
      usingDataMode:(GKSendDataMode)mode
{
    [ self.mySession sendDataToAllPeers:data        // NSData *
                           withDataMode:mode        // reliable or unreliable
                                  error:nil ];
}
```

```
- (void)receiveData:(NSData *)data
           fromPeer:(NSString *)peer
          inSession:(GKSession *)session
            context:(void *)context
{
    // unpack the NSData, update game state
}
```

Peer-to-Peer Connection

5. Handle GKSession state changes

```
- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state
{
    switch (state)
    {
        case GKPeerStateConnected:
          // ...
          break;

        case GKPeerStateDisconnected:
          // handle disconnection
          break;

        ...
    }
}
```

Peer-to-Peer Connection

5. Handle GKSession state changes

```
- (void)session:(GKSession *)session
           peer:(NSString *)peerID
 didChangeState:(GKPeerConnectionState)state
{
    switch (state)
    {
        case GKPeerStateConnected:
          // ...
          break;

        case GKPeerStateDisconnected:
          // handle disconnection
          break;

        ...
    }
}
```

Peer-to-Peer Connection

Steps to connect over bluetooth

1. Configure and present GKPeerPickerController
2. Provide a GKSession
3. Handle picker connecting to peer (or user cancel)
4. Implement methods for GKSession to send and receive data
5. Handle GKSession state changes

In-Game Voice

GKVoiceChatService

Adding in-game voice chat

- Allows voice chat between devices
  - Samples your microphone and sends audio to remote participant
  - Plays audio received from the remote participant
  - Supports mixing game audio with in-game voice

- Your application provides the initial handshake
  - Pass configuration data to initiate chat

- Audio monitoring and adjustments
  - Microphone mute, remote volume, local and remote metering

GKVoiceChatService

Sequence of events

- Both sides define unique participant ID

```
- (NSString *)participantID;
```

- You initiate the chat with GKVoiceChatService

```
- (BOOL)startVoiceChatWithParticipantID:(NSString *)participantID
                                 error:(NSError **)error;
```

Local

- GKVoiceChatClient delegate fires

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
                sendData:(NSData *)data
         toParticipantID:(NSString *)participantID;
```

Local

- You send this data to the remote participant

GKVoiceChatService

Sequence of events

- Remote instance receives data from network
  - Identifies as voice chat configuration data, invokes:

Remote

```
[ myVoiceChatClient receivedData:data fromParticipantID:participantID ];
```

- GKVoiceChatService delegate fires

Remote

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
        didReceiveInvitationFromParticipantID:(NSString *)participantID
        callID:(NSInteger)callID
{
    // Allow user to accept / deny chat invitation

    // If accepted
    [ myVoiceChatService acceptCallID:callID error:&err ];
}
```

GKVoiceChatService

Sequence of events

- Connection is negotiated
- Both sides receive notification (optional)

Local

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
        didStartWithParticipantID:(NSString *)participantID
```

Remote

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
        didStartWithParticipantID:(NSString *)participantID
```

Agenda

- Part 1
  - Welcome and Overview — Done
  - Graphics — Done
  - Audio — Done
- Part 2
  - Peer-to-Peer Connectivity — Done
  - Accelerometer — 20 minutes
  - In-App Purchase — 20 minutes

Accelerometer

Force Input

Rotate, pivot, spin, tilt, and shake

- Built-in accelerometer
  - Reports how the device is oriented
  - Instantaneous knowledge of movements
- Input to a game
  - Steer
  - Throttle, brake
  - Swing, punch, throw
- Design unique input methods
  - Flip to “jump”
  - Shake to “reload”

Accelerometer

What you'll learn

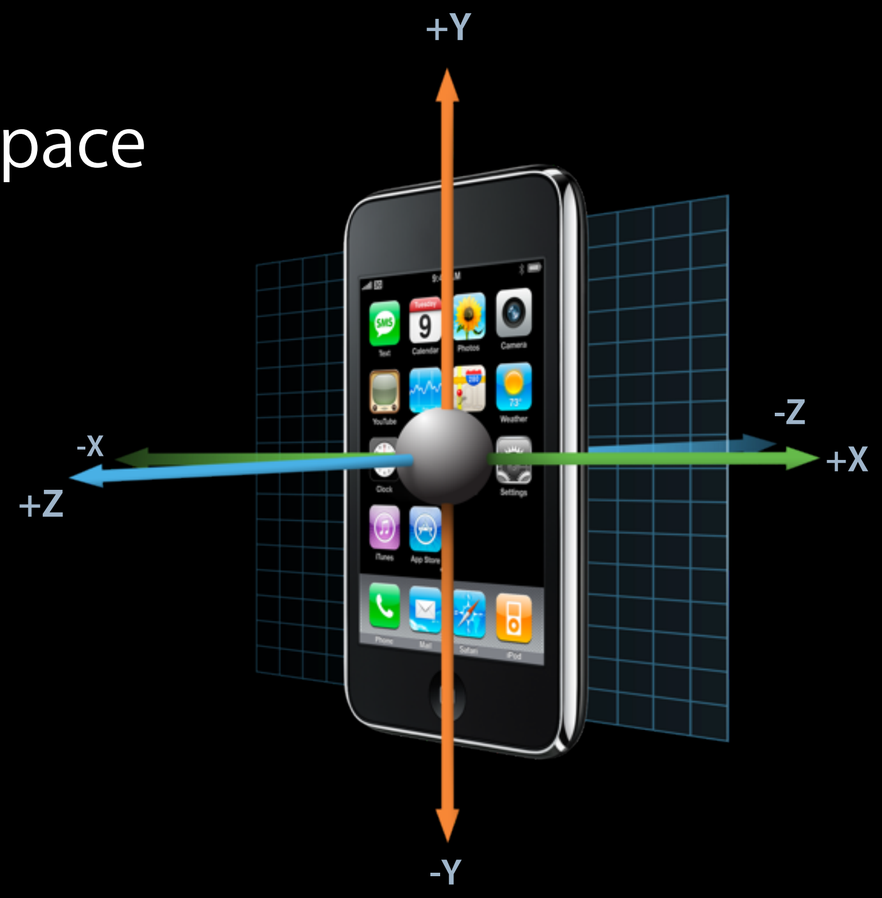
- The interface
  - Configuring the accelerometer
  - Retrieving accelerometer data
- Smoothing the data
- User experience tips

No Simulator Support

The Accelerometer Interface

Three-axis data

- Part of the UIKit framework
- Three axes within a three-dimensional space
- Facing you in portrait orientation
  - X-axis runs horizontally, left to right
  - Y-axis runs vertically, bottom to top
  - Z-axis runs through, back to front



Configuring the Accelerometer

Starting the event delivery

```
- (void)enableAccelerometerEvents
{
    UIAccelerometer* theAccel = [UIAccelerometer sharedAccelerometer];

    theAccel.updateInterval = 1.0f / kFrequency;  // max 100 Hz

    theAccel.delegate = self;
}
```

Configuring the Accelerometer

Starting the event delivery

```
- (void)enableAccelerometerEvents
{
    UIAccelerometer* theAccel = [UIAccelerometer sharedAccelerometer];

    theAccel.updateInterval = 1.0f / kFrequency;  // max 100 Hz

    theAccel.delegate = self;
}
```

Configuring the Accelerometer

Starting the event delivery

```
- (void)enableAccelerometerEvents
{
    UIAccelerometer* theAccel = [UIAccelerometer sharedAccelerometer];

    theAccel.updateInterval = 1.0f / kFrequency;  // max 100 Hz

    theAccel.delegate = self;
}
```

Event Delivery Begins as Soon as you assign the Delegate

```
- (void)accelerometer:(UIAccelerometer*)accelerometer
        didAccelerate:(UIAcceleration*)acceleration
```

Defining Your Delegate Object

Retrieving the accelerometer data

```
- (void)accelerometer:(UIAccelerometer*)accelerometer
        didAccelerate:(UIAcceleration*)acceleration
{
    // Get the event data
    UIAccelerationValue  x, y, z;

    x = acceleration.x;
    y = acceleration.y;
    z = acceleration.z;

    // Process the data...
}
```

Defining Your Delegate Object

Retrieving the accelerometer data

```
- (void)accelerometer:(UIAccelerometer*)accelerometer
        didAccelerate:(UIAcceleration*)acceleration
{
    // Get the event data
    UIAccelerationValue  x, y, z;

    x = acceleration.x;
    y = acceleration.y;
    z = acceleration.z;

    // Process the data...
}
```

Defining Your Delegate Object

Retrieving the accelerometer data

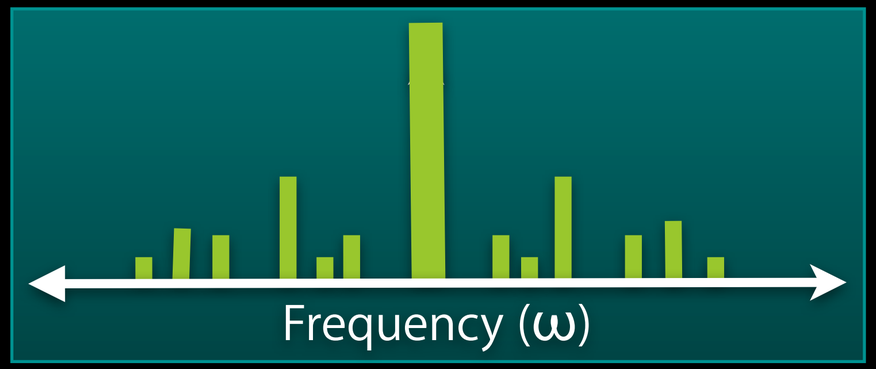
```
- (void)accelerometer:(UIAccelerometer*)accelerometer
        didAccelerate:(UIAcceleration*)acceleration
{
    // Get the event data
    UIAccelerationValue  x, y, z;

    x = acceleration.x;
    y = acceleration.y;
    z = acceleration.z;

    // Process the data...
}
```

Processing Accelerometer Data

Accelerometer is VERY sensitive

NOTE: values multiplied 50x

f(t)



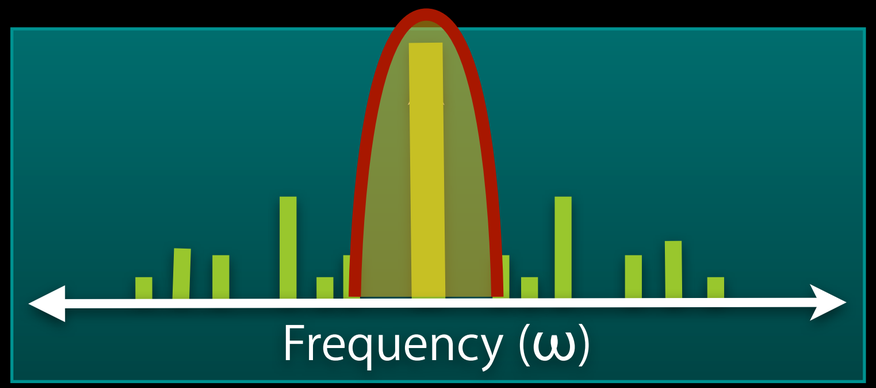
F(ω)

Processing Accelerometer Data

Using filters to isolate data components

NOTE:
values
multiplied
50x

f(t)



Low-pass filter
isolates “gravity”

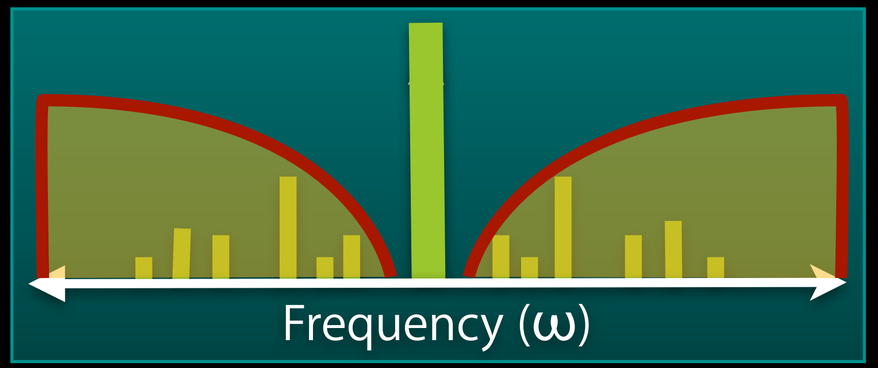
F(ω)

Processing Accelerometer Data

Using filters to isolate data components

NOTE:
values
multiplied
50x

f(t)



High-pass filter
isolates “shake”

F(ω)

Processing Accelerometer Data

Applying a low-pass filter

- Trivial low-pass filter example

```
#define FILTERFACTOR  0.1

value = (newAcceleration * FILTERFACTOR) +
        (previousValue * (1.0 - FILTERFACTOR));

previousValue = value;
```

OK for simple orientation

NOT good enough for a game controller

Processing Accelerometer Data

Applying a low-pass filter

- Trivial low-pass filter example

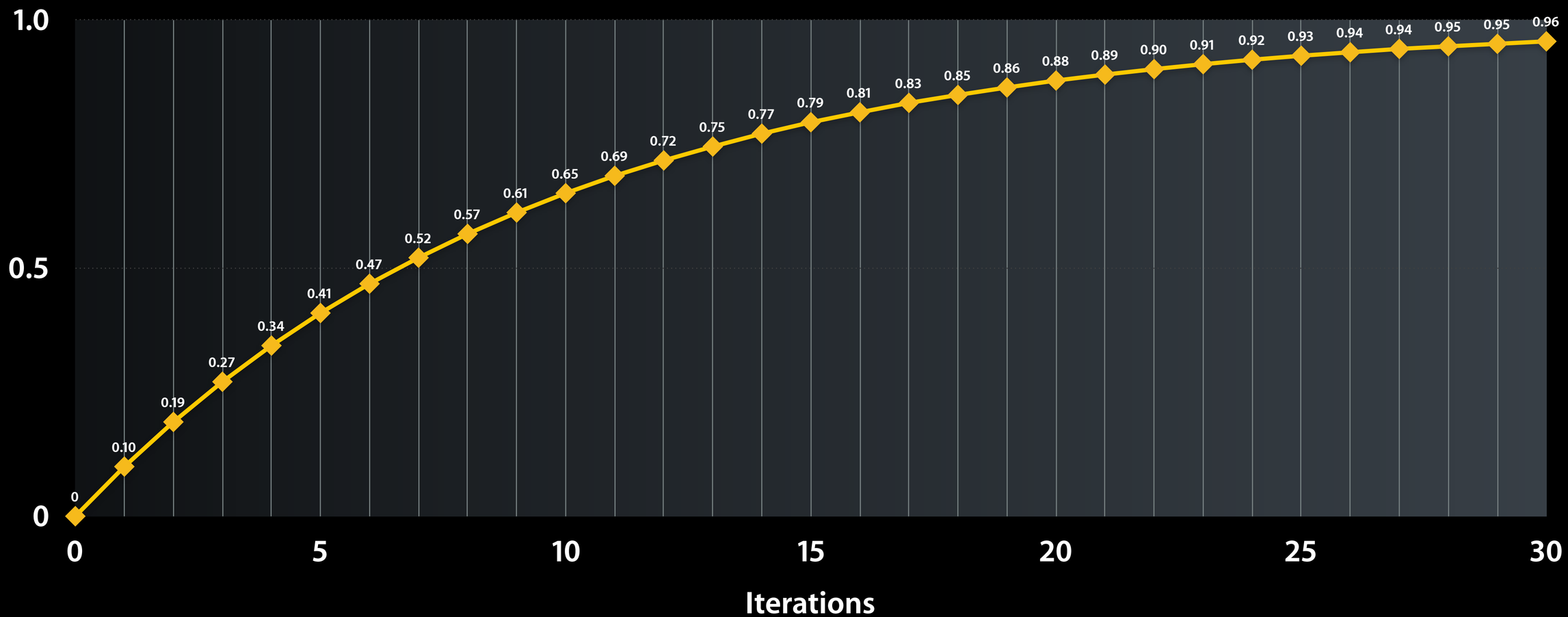
```
value = (newAcceleration * 0.1) + (previousValue * 0.9);


previousValue = value;
```

OK for simple orientation

NOT good enough for a game controller

Trivial Low-Pass Filter

- Start value = 0.0
- New value = 1.0



Processing Accelerometer Data

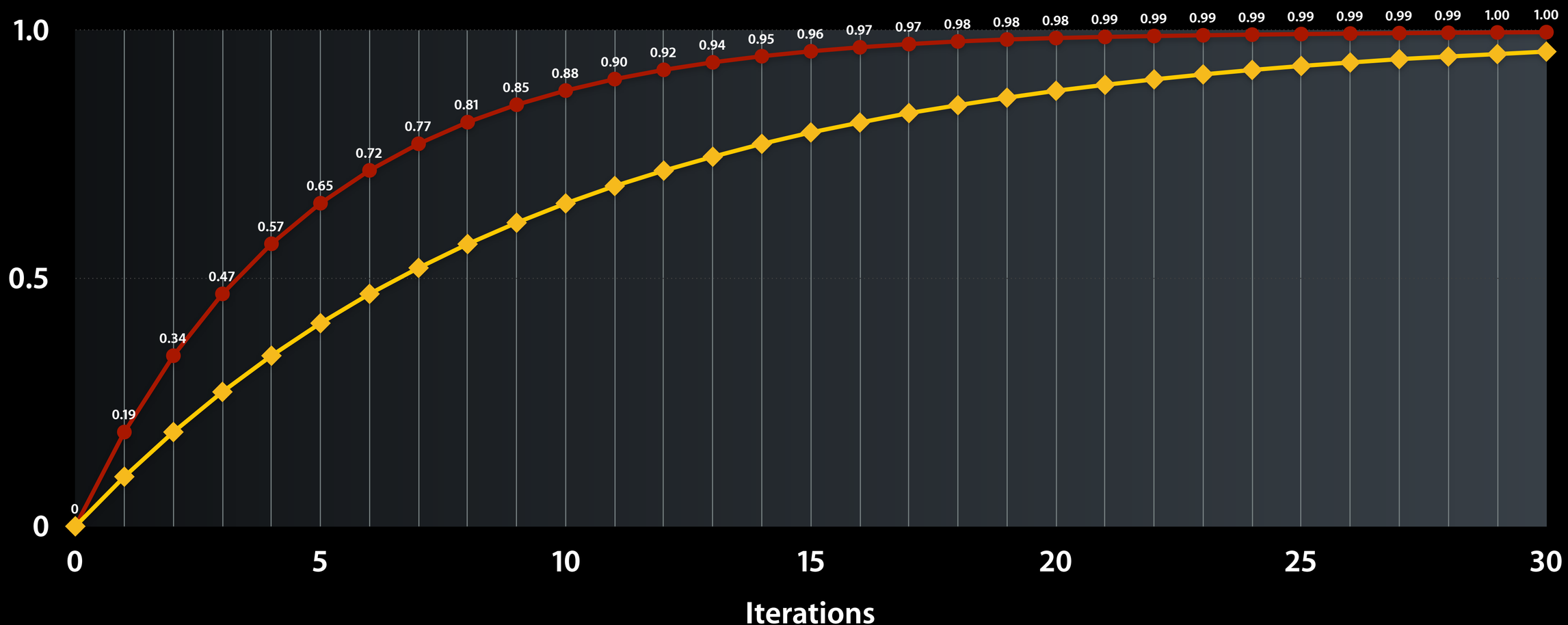
Better low-pass filter

```
#define FILTERFACTOR 0.1
#define ATTENUATION  3.0
#define MINSTEP      0.02

// Start with the value from a basic low-pass filter
basevalue = (newAcceleration * FILTERFACTOR) + (previousValue * (1.0 - FILTERFACTOR));

// Smooth more for small movement to filter out noise
d = Clamp(ABS(Normalize(basevalue)-Normalize(newAcceleration))/MINSTEP - 1., 0., 1.);

// Calculate a proportional factor
alpha = (1.-d) * FILTERFACTOR / ATTENUATION + d * FILTERFACTOR;

// Use the new factor to offset smoothed values
value = newAcceleration * alpha + basevalue * (1. - alpha);
```

Available in AccelerometerGraph

Better Low-Pass Filter

- Start value = 0.0
- New value = 1.0

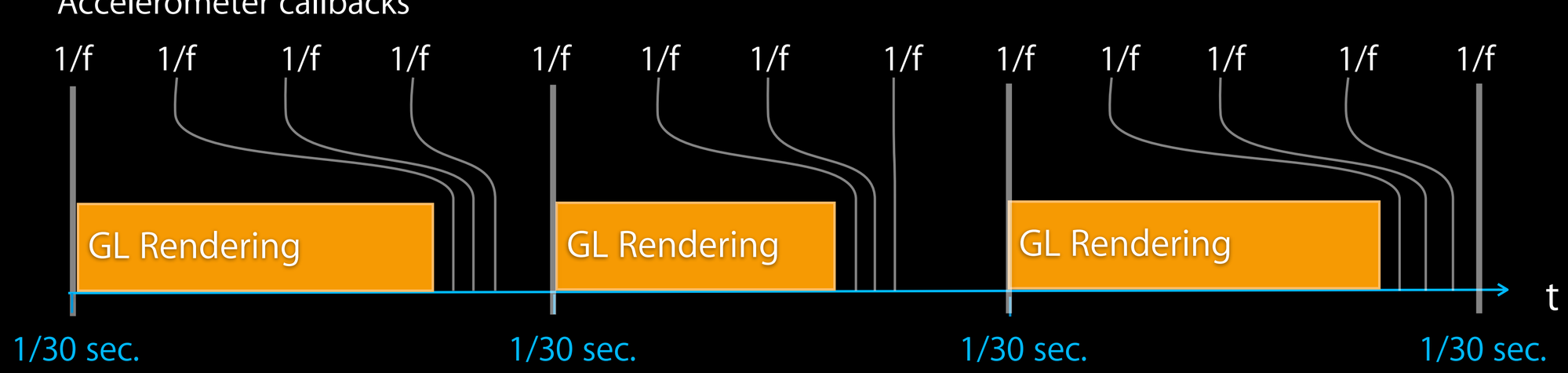


Demo

GLGravity

Expert Tip

- Callback is in main thread
  - So, processing frequency may be irregular



- Perform smoothing in rendering callbacks instead of accelerometer callbacks

Accelerometer "Home" Position

Choose a home (zero) position

- Provide users with a naturally comfortable orientation
  - Choose something reasonable?
    - (e.g. 30 degree incline)
  - Calibrate at startup?
    - (and pause/restart)
  - No universal answer

Disable Screen “Dimming”

Full-screen gaming tips

- Important for accelerometer-based games
- UIApplication property

```
[UIApplication sharedApplication].idleTimerDisabled = YES;
```

- Notes:
  - Set to YES during gameplay only
  - Set to NO during menu browsing, info screens, etc.

Agenda

- Part 1
  - Welcome and Overview
  - Graphics
  - Audio
- Part 2
  - Peer-to-Peer Connectivity
  - Accelerometer
  - In-App Purchase

- Done
- Done
- Done
- Done
- Done
- 20 minutes

In-App Purchase

In-App Purchase

Store Kit

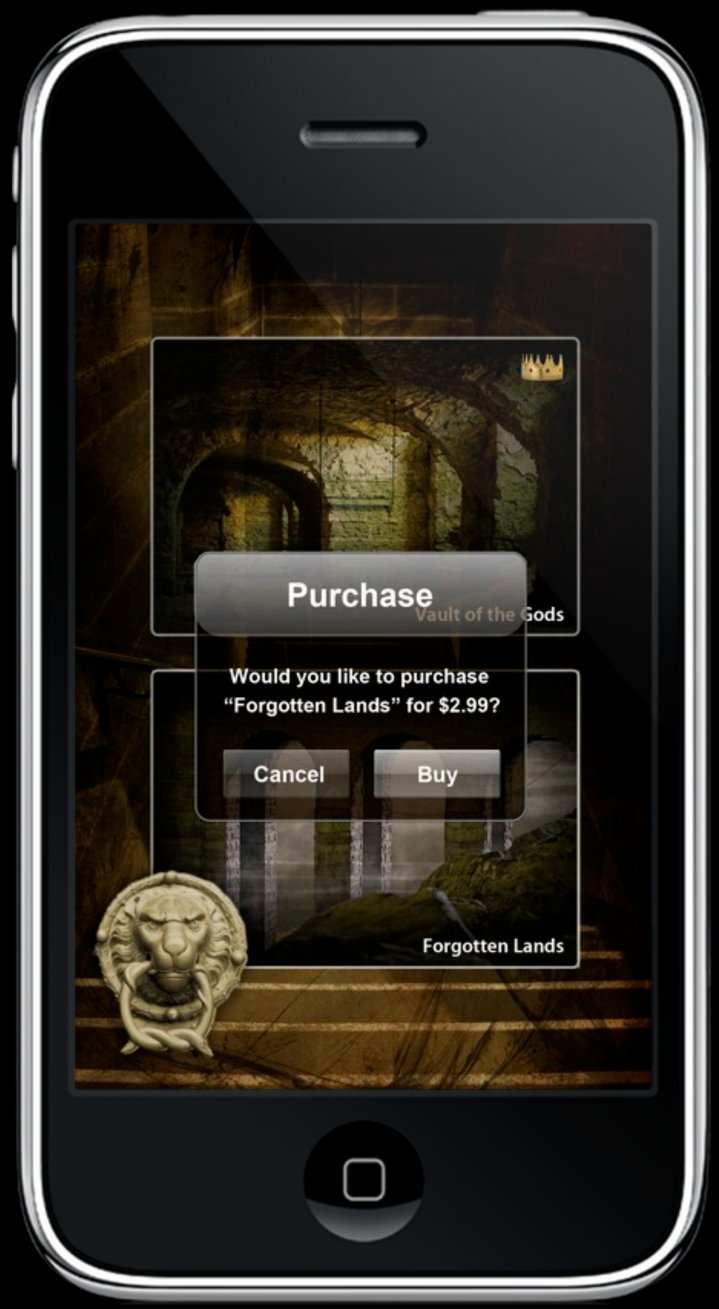
- Payment collection engine for iPhone OS
  - Requires iPhone OS 3.0
  - For paid apps only
- Facilitates In-App purchases
  - Unlock game features
  - Access additional content
- App Store collects the payment
  - Same purchase experience and receipt
  - Updated iTunes Connect reports

In-App Purchase

Scenario for today…

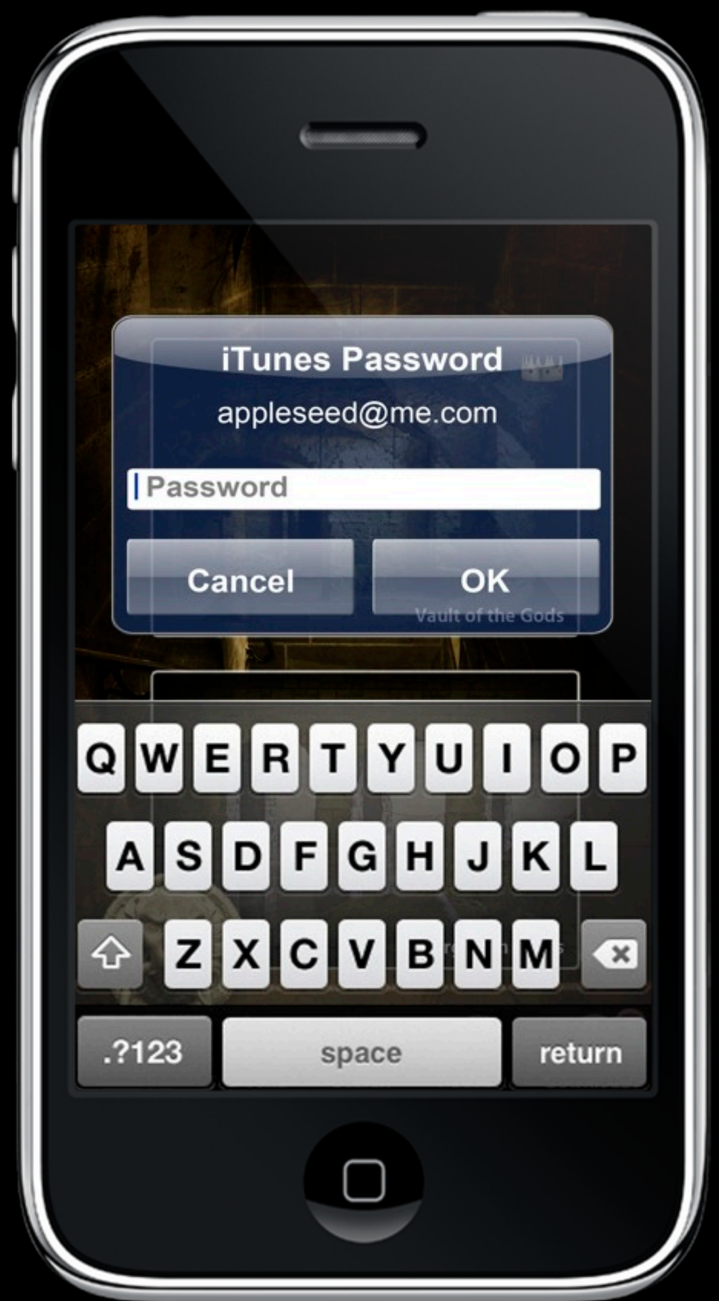
Usage Scenario

Unlock levels



Usage Scenario

Unlock levels



Usage Scenario

Unlock levels

Implementing In-App Purchase

- Up front
  - Define Metadata
  - Enter into iTunes Connect
- Build purchase workflow
  - Presentation
  - Selection
  - Delivery
  - Restoration

Define Metadata

SKU	Title	Description	Price Tier
DG01	“Assault of the unknown”	“A dungeon adventure…”	Tier 3
DG02	“Vault of the Gods”	“Explore the temple… ”	Tier 3
DG03	“Beyond the Darkness”	“Descend into the mines…”	Tier 3
DG04	“Forgotten Lands”	“New worlds campaign…”	Tier 3
…	…	…	…

Upload to iTunes Connect

Lookup product list

iTunes Connect

DG01	“Assault of the unknown”	“A dungeon adventure…”	Tier 3
DG02	“Vault of the Gods”	“Explore the temple… ”	Tier 3
DG03	“Beyond the Darkness”	“Descend into the mines…”	Tier 3
DG04	“Forgotten Lands”	“New worlds campaign…”	Tier 3
…	…	…	…

StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

Load Catalog

- Look up product list
  - From hardcoded values
  - From plist
  - From server (dynamic)

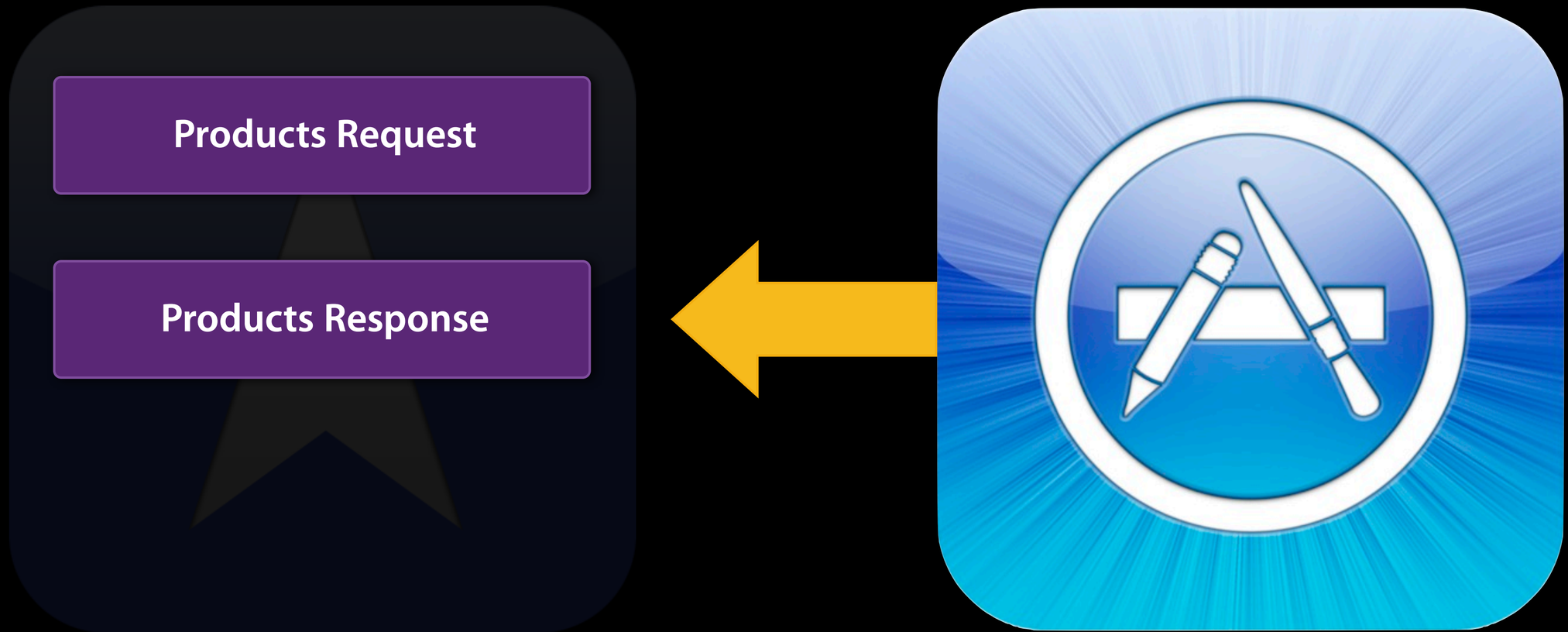
Load Catalog

- Look up product list
  - From hardcoded values
  - From plist
  - From server (dynamic)

Fetch Localized Details

```
products = [ NSSet setWithObjects: @”DG01”, @”DG02”, @”DG03”, @”DG04”, nil ]
request = [ [ SKProductsRequest alloc ] initWithProductIdentifiers:products ];
request.delegate = self;

[ request start ];
```

Fetch Localized Details



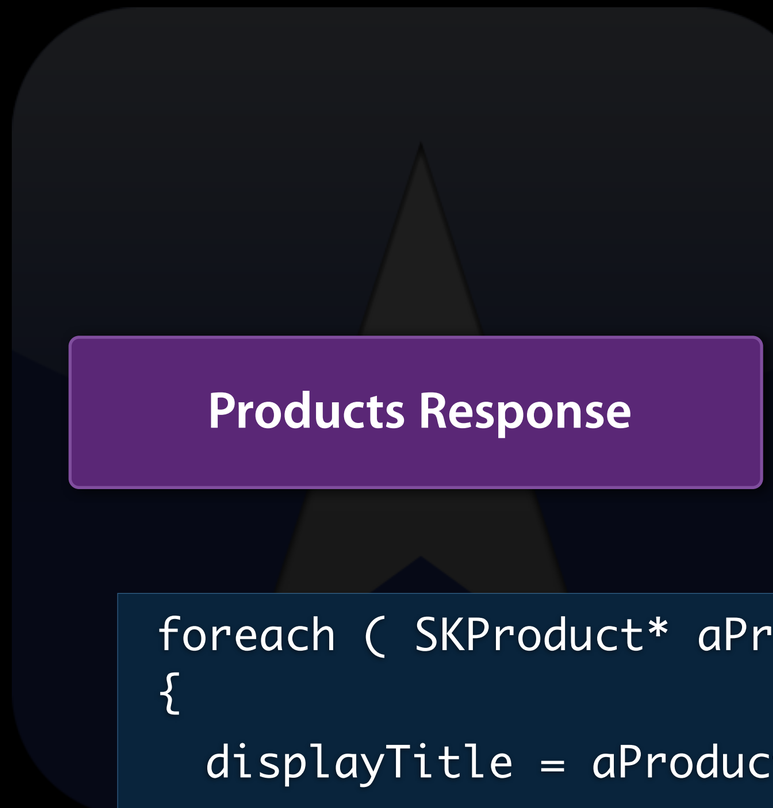

```
-(void)productsRequest:(SKProductsRequest *)request
    didReceiveResponse:(SKProductsResponse *)response
```

Fetch Localized Details

```
response.products

response.invalidProductIdentifiers
```

- Invalid Products
  - Never entered into iTunes Connect
  - Aren’t marked as “Cleared for Sale”
  - Haven’t propagated to all servers

Fetch Localized Details



response.products

response.invalidProductIdentifiers

```
foreach ( SKProduct* aProduct in response.products )
{
  displayTitle = aProduct.localizedTitle;
  displayDesc  = aProduct.localizedDescription;
  displayPrice = aProduct.price;
  // present to user
}
```

Present Items for Selection

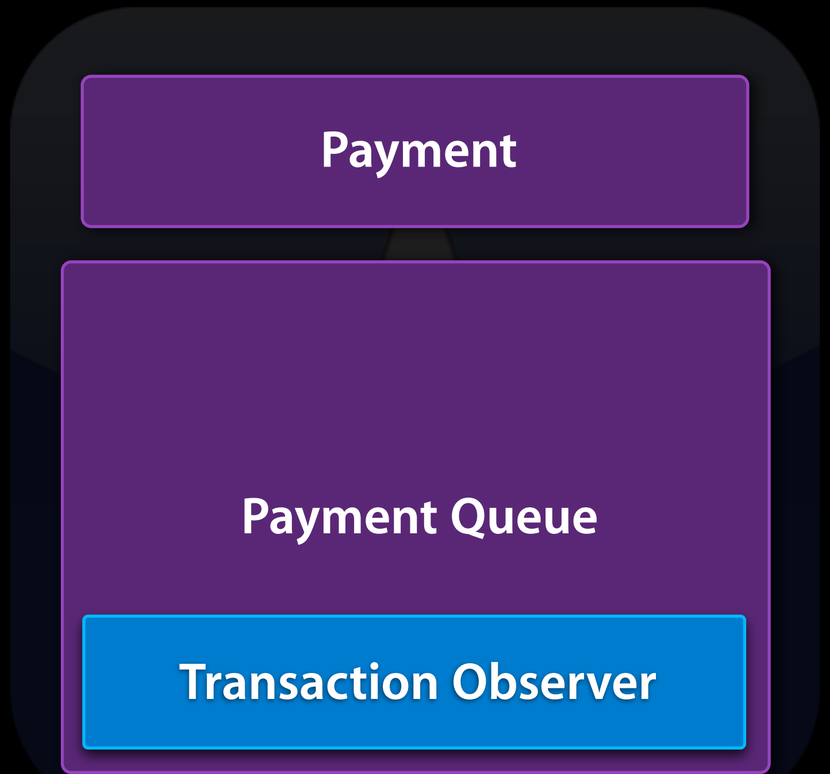
StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

Request Item Payment

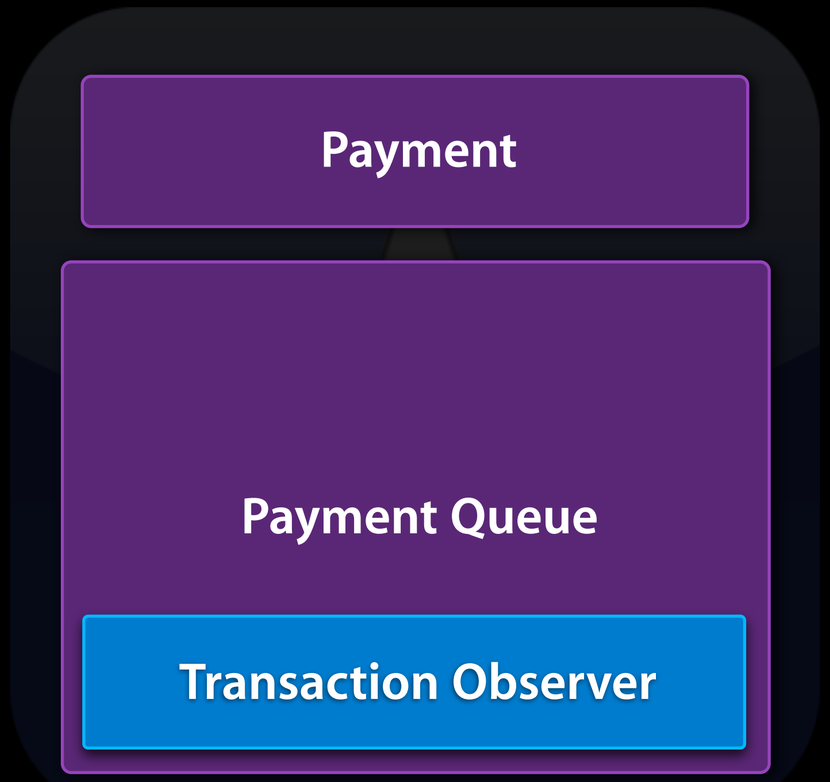
```
myPayment = [ SKPayment paymentForProduct:selectedProduct userInfo:nil ]

                                    or

myPayment = [ SKPayment paymentForProductIdentifier:selectedProdID userInfo:nil ];
```

Request Item Payment



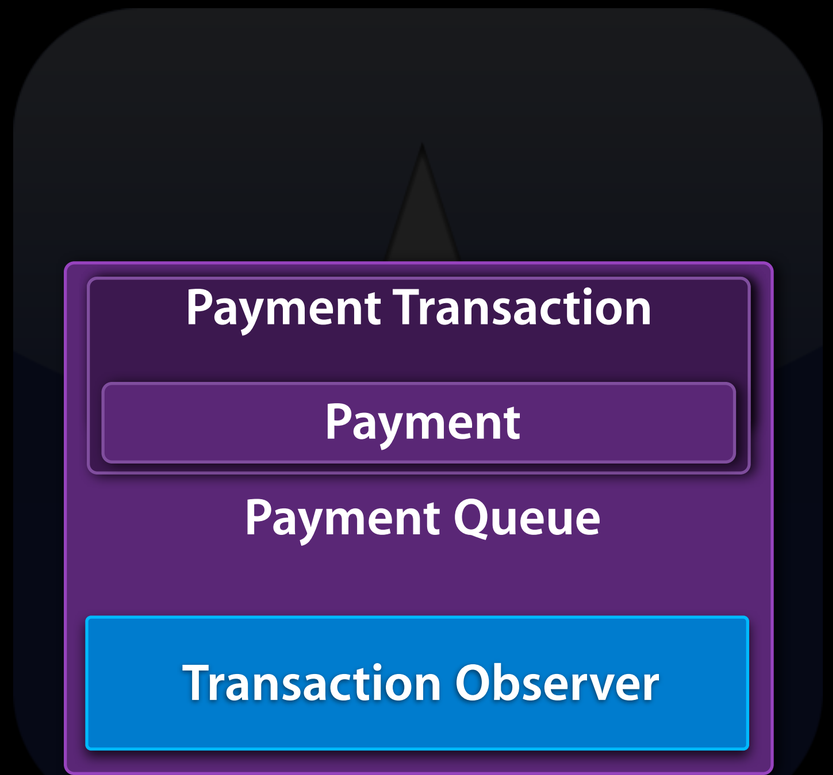

```
defaultPaymentQueue = [ SKPaymentQueue defaultQueue ];
[ defaultPaymentQueue addTransactionObserver:myObserver ];
```

Request Item Payment



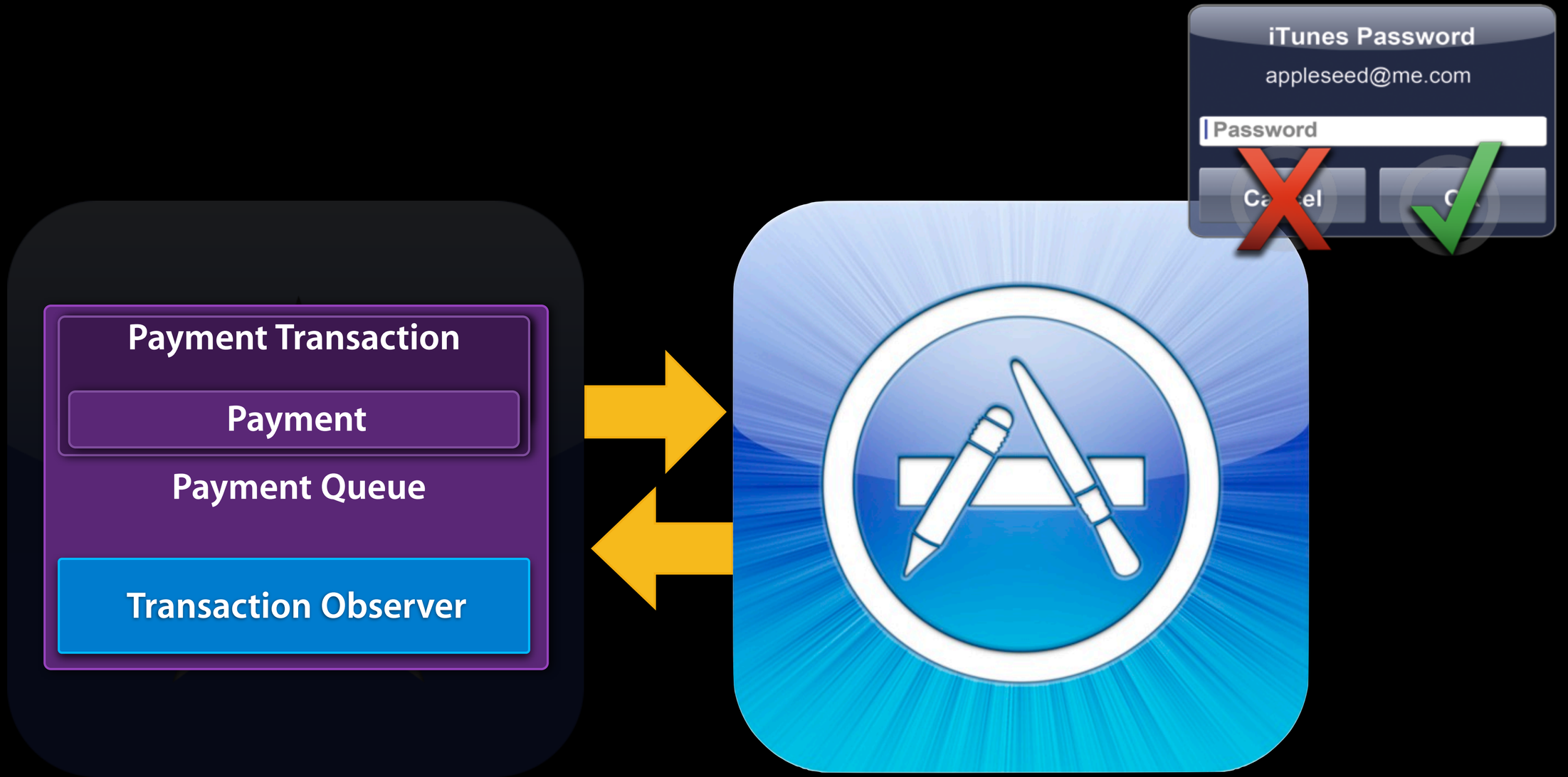

```
defaultPaymentQueue = [ SKPaymentQueue defaultQueue ];
[ defaultPaymentQueue addTransactionObserver:myObserver ];
[ defaultPaymentQueue addPayment:myPayment ];
```

Request Item Payment

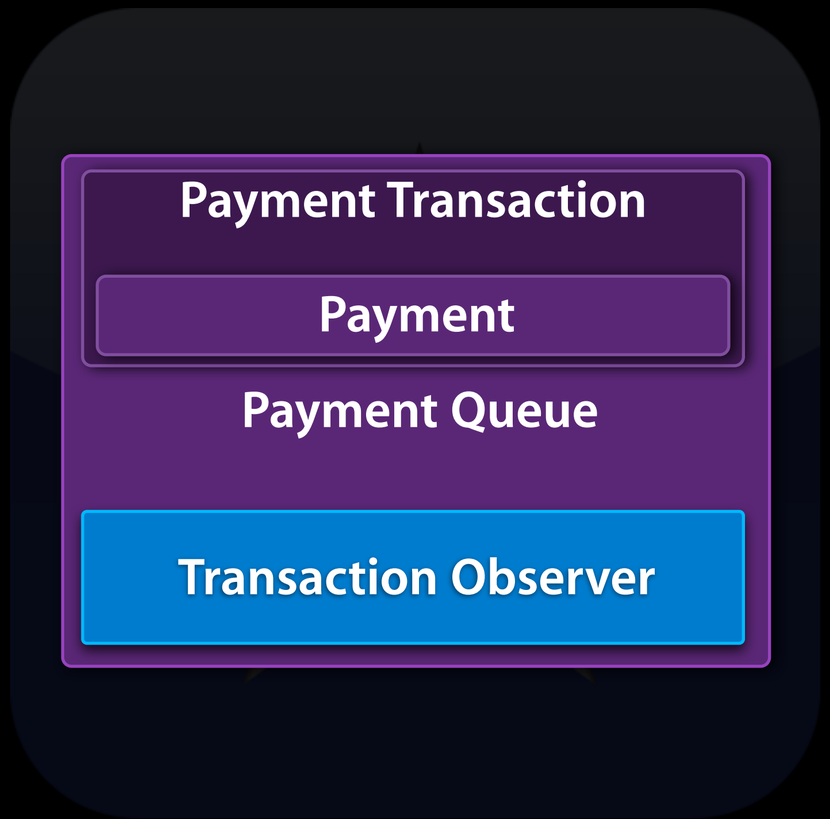



```
defaultPaymentQueue = [ SKPaymentQueue defaultQueue ];
[ defaultPaymentQueue addTransactionObserver:myObserver ];
[ defaultPaymentQueue addPayment:myPayment ];
```

Request Item Payment



Request Item Payment




```
- (void)paymentQueue:(SKPaymentQueue *)queue
  updatedTransactions:(NSArray *)transactions
```

StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

Inspect Payment Transaction

```
for(SKPaymentTransaction *trans in transactions) {
    switch (trans.transactionState ) {

  case SKPaymentTransactionStatePurchasing:
      // item is in the process of being purchased
      break;

   case SKPaymentTransactionStateFailed:
       // user canceled the transaction
       break;

  case SKPaymentTransactionStatePurchased:
      // item was successfully purchased
      // UNLOCK THE PURCHASE...
      break;

 case SKPaymentTransactionStateRestored:
      // already purchased this product
      // UNLOCK THE PURCHASE...
      break;

...
```

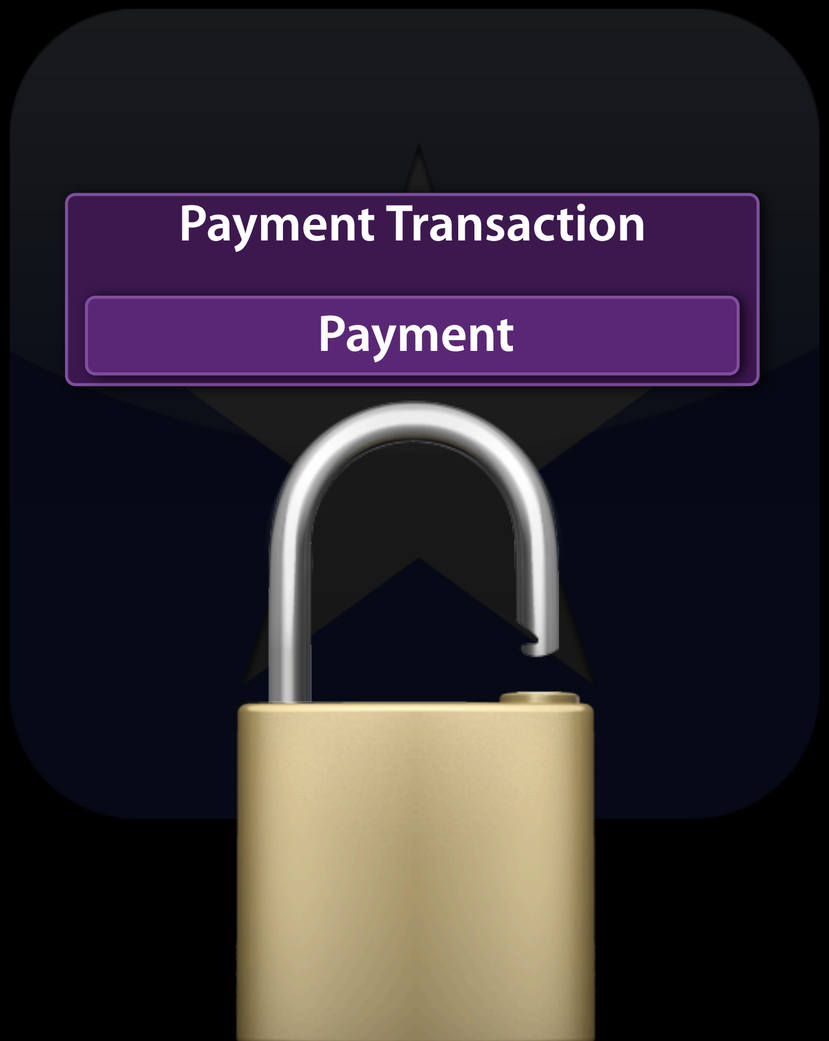
Inspect Payment Transaction

```
for(SKPaymentTransaction *trans in transactions) {
    switch (trans.transactionState ) {

  case SKPaymentTransactionStatePurchasing:
      // item is in the process of being purchased
      break;

   case SKPaymentTransactionStateFailed:
       // user canceled the transaction
       break;

  case SKPaymentTransactionStatePurchased:
      // item was successfully purchased
      // UNLOCK THE PURCHASE...
      break;

 case SKPaymentTransactionStateRestored:
      // already purchased this product
      // UNLOCK THE PURCHASE...
      break;

...
```

Inspect Payment Transaction

```
for(SKPaymentTransaction *trans in transactions) {
    switch (trans.transactionState ) {

  case SKPaymentTransactionStatePurchasing:
      // item is in the process of being purchased
      break;

   case SKPaymentTransactionStateFailed:
      // user canceled the transaction
      break;

  case SKPaymentTransactionStatePurchased:
      // item was successfully purchased
      // UNLOCK THE PURCHASE...
      break;

   case SKPaymentTransactionStateRestored:
      // already purchased this product
      // UNLOCK THE PURCHASE...
      break;

...
```

Inspect Payment Transaction




```
for(SKPaymentTransaction *trans in transactions) {
    switch (trans.transactionState ) {

  case SKPaymentTransactionStatePurchasing:
      // item is in the process of being purchased
      break;

   case SKPaymentTransactionStateFailed:
      // user canceled the transaction
      break;

  case SKPaymentTransactionStatePurchased:
      // item was successfully purchased
      // UNLOCK THE PURCHASE...
      break;

   case SKPaymentTransactionStateRestored:
      // already purchased this product
      // UNLOCK THE PURCHASE...
      break;

...
```

Processing the Transaction Queue

“Finishing” the transaction

- Users may quit your App anytime
- Store Kit Payment Queue is persistent
- Transactions are persistent
- Confirm access before finishing transaction

```
[ [ SKPaymentQueue defaultQueue ] finishTransaction:transaction ];
```

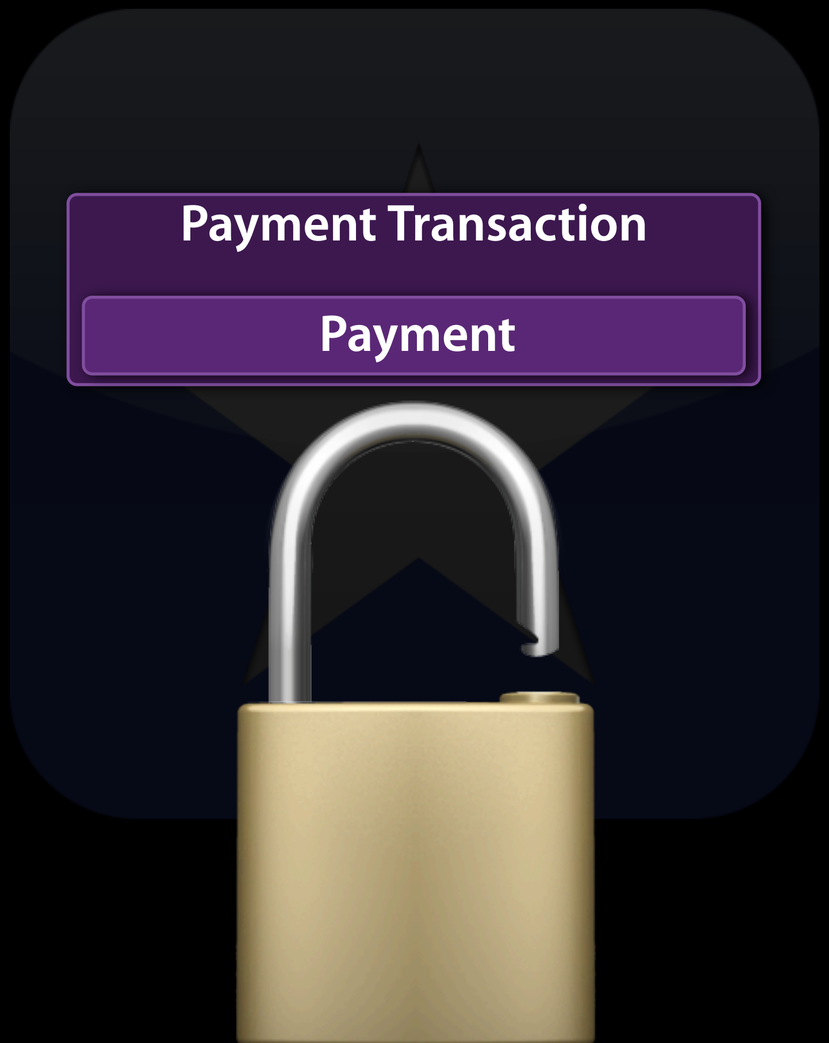
StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

Support Item Restore

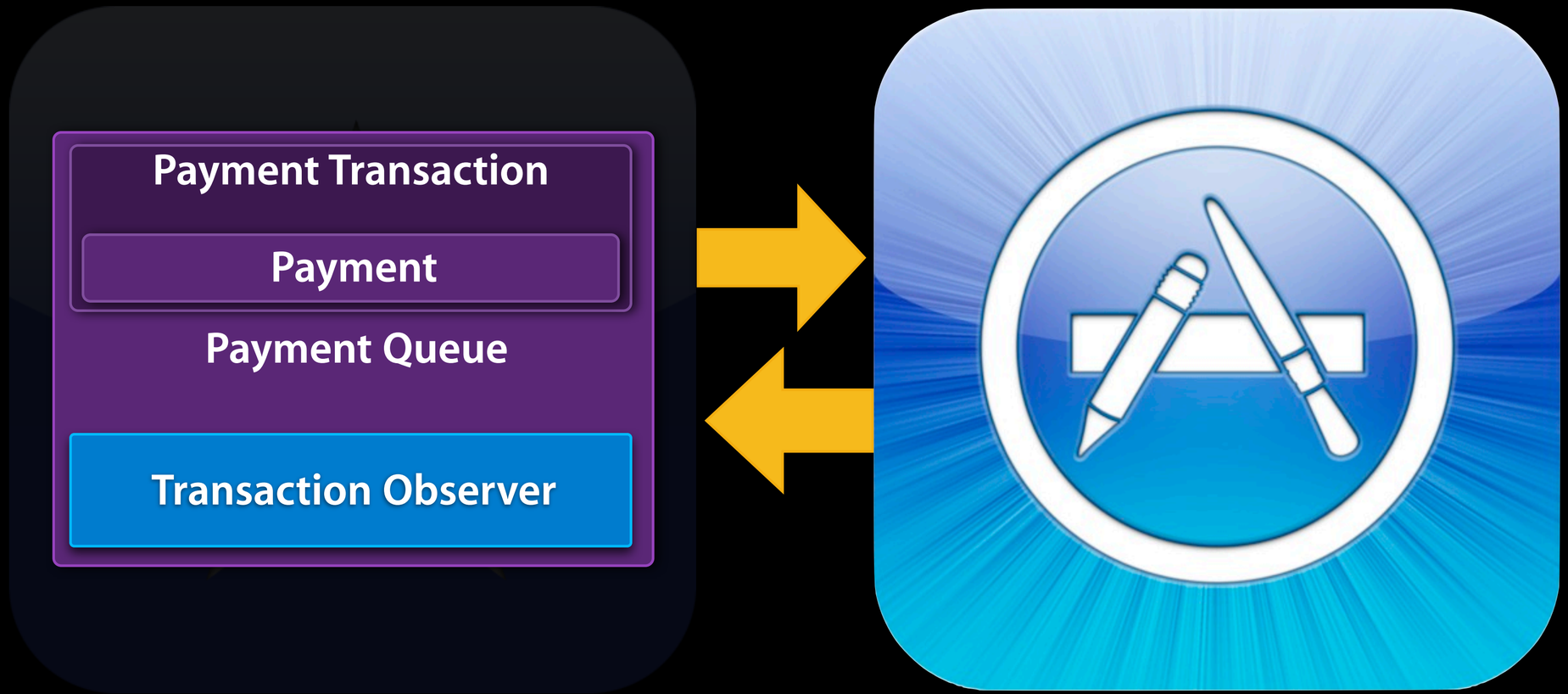
- Like apps, Purchased Items are only paid for once
- Available across customer devices
- Available after app is deleted then reinstalled

Support Item Restore



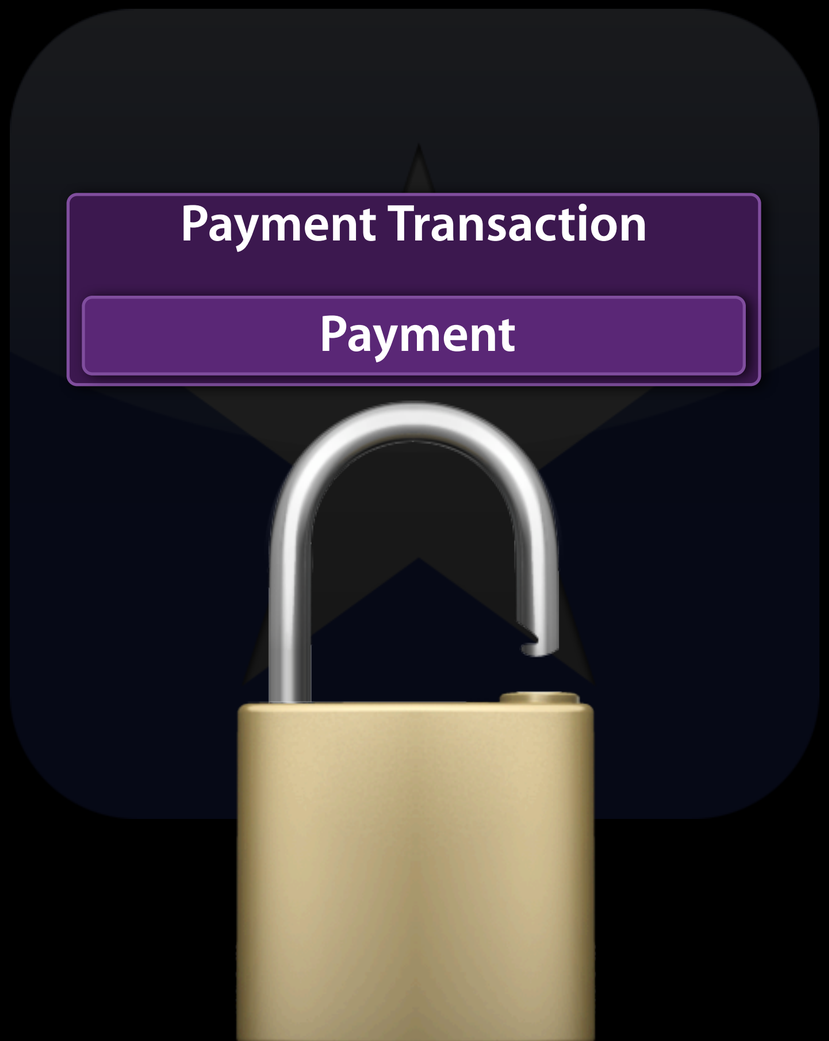

```
for(SKPaymentTransaction *trans in transactions) {
    switch (trans.transactionState ) {

  case SKPaymentTransactionStatePurchasing:
      // item is in the process of being purchased
      break;

   case SKPaymentTransactionStateFailed:
      // user canceled the transaction
      break;

  case SKPaymentTransactionStatePurchased:
      // item was successfully purchased
      // UNLOCK THE PURCHASE...
      break;

   case SKPaymentTransactionStateRestored:
      // already purchased this product
      // UNLOCK THE PURCHASE...
      break;

...
```

Support Item Restore




```
defaultPaymentQueue = [ SKPaymentQueue defaultQueue ];
[ defaultPaymentQueue restoreCompletedTransactions ];
```

Support Item Restore




```
for(SKPaymentTransaction *trans in transactions) {
    switch (trans.transactionState ) {

  case SKPaymentTransactionStatePurchasing:
      // item is in the process of being purchased
      break;

   case SKPaymentTransactionStateFailed:
      // user canceled the transaction
      break;

  case SKPaymentTransactionStatePurchased:
      // item was successfully purchased
      // UNLOCK THE PURCHASE...
      break;

   case SKPaymentTransactionStateRestored:
      // already purchased this product
      // UNLOCK THE PURCHASE...
      break;

...
```

StoreKit Integration

- Prior to purchase
  - Load catalog
  - Fetch localized details
  - Present items for selection
- User selects item
  - Request item payment
- After transaction
  - Inspect payment transaction
  - Unlock built-in items
  - “Finish” the transaction
- Restoration
  - Support item restore

Agenda

- Part 1
  - Welcome and Overview — Done
  - Graphics — Done
  - Audio — Done
- Part 2
  - Peer-to-Peer Connectivity — Done
  - Accelerometer — Done
  - In-App Purchase — Done

Related Sessions

Audio Development for iPhone OS	Presidio Tuesday 3:30PM
Audio Processing and Playback	Presidio Tuesday 5:00PM
Peer to Peer Networking with Game Kit	Presidio Wednesday 9:00AM
In-App Purchase on iPhone	Presidio Wednesday 10:30AM
OpenGL ES Overview for iPhone OS	Presidio Wednesday 2:00PM
OpenGL ES Shading and Rendering	Presidio Wednesday 3:30PM
OpenGL ES Tuning and Techniques	Presidio Wednesday 5:00PM
Accessing the iPod Library	Pacific Heights Thursday 5:00PM

Lab

Game Development for iPhone Lab	iPhone Lab C Thursday 9:00AM

More Information

Allan Schaffer
Graphics Technology Evangelist
aschaffer@apple.com

Documentation
iPhone OS Programming Guide
http://developer.apple.com/iPhone

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09